Чисто виртуальные функции. Раннее и позднее связывание, механизм позднего связывания

Чисто виртуальные функции и абстрактные типы

Когда виртуальная функция не переопределена в производном классе, то при вызове ее в объекте производного класса вызывается версия из базового класса. Однако во многих случаях невозможно ввести содержательное определение виртуальной функции в базовом классе. Например, при объявлении базового класса figure в предыдущем примере определение функции show_area() не несет никакого содержания. Она не вычисляет и не выводит на экран площадь объекта какого- либо типа. Имеется два способа действий в подобной ситуации. Первый, подобно предыдущему примеру, заключается в выводе какого-нибудь предупреждающего сообщения. Хотя такой подход полезен в некоторых ситуациях, он не является универсальным. Могут быть такие виртуальные функции, которые обязательно должны быть определены в производных классах, без чего эти производные классы не будут иметь никакого значения. Например, класс triangle бесполезен, если не определена функция show_area(). В таких случаях необходим метод, гарантирующий, что производные классы действительно определят все необходимые функции. Язык C++ предлагает в качестве решения этой проблемы чисто виртуальные функции.

Чисто виртуальная функция (pure virtual function) является функцией, которая объявляется в базовом классе, но не имеет в нем определения. Поскольку она не имеет определения, то есть тела в этом базовом классе, то всякий производный класс обязан иметь свою собственную версию определения. Для объявления чисто виртуальной функции используется следующая общая форма:

virtual тип имя_функции(список параметров) = 0;

Здесь тип обозначает тип возвращаемого значения, а имя_функции является именем функции. Например, следующая версия функции show_area() класса figure является чисто виртуальной функцией.

```
class figure {
double x, y;
public:
void set_dim(double i, double j=0) {
x = i;
y = j;
}
virtual void show_area() = 0; // чисто виртуальная
};
```

При введении чисто виртуальной функции в производном классе обязательно необходимо определить свою собственную реализацию этой функции. Если класс не будет содержать определения этой функции, то компилятор выдаст ошибку. Например, если попытаться откомпилировать следующую модифицированную версию программы figure, в которой удалено определение функции snow_area() из класса circle, то будет выдано сообщение об ошибке:

```
/* Данная программа не компилируется, поскольку класс circle не переопределил
show_area() */
#include <iostream.h>
class figure {
protected:
double x, y;
```

```
public:
void set_dim(double i, double j) {
x = i;
y = j;
}
virtual void show_area() = 0; // pure
};
class triangle: public figure {
public:
void show_area() {
cout << "Triangle with height ";
cout << x << " and base " << y;
cout << " has an area of ";
cout << x * 0.5 * y << ". \ n";
}
};
class square: public figure {
public:
void show_area() {
cout << "Square with dimensions ";
cout << x << "x" << y;
cout << " has an area of ";
cout << x * y << " . \n";
}
};
class circle: public figure {
// определение show_area() отсутствует и потому выдается ошибка
};
int main ( )
{
figure *p; // создание указателя базового типа
circle c; // попытка создания объекта типа circle - ОШИБКА
triangle t; // создание объектов порожденных типов
square s;
p = &t;
p->set_dim(10.0, 5.0);
p->show_area ();
p = &s;
p->set_dim(10.0, 5.0);
p->show_area();
return 0;
}
```

Если какой-либо класс имеет хотя бы одну чисто виртуальную функцию, то такой класс называется *абстрактным* (abstract). Важной особенностью абстрактных классов является то, что не существует ни одного объекта данного класса. Вместо этого абстрактный класс служит в качестве базового для других производных классов. Причина, по которой абстрактный класс не может быть использован для объявления объекта, заключается в том, что одна или несколько его функций-членов не имеют определения. Тем не менее, даже если базовый класс является абстрактным, все равно можно объявлять указатели или ссылки на него, с помощью которых затем поддерживается полиморфизм времени исполнения.

Раннее и позднее связывание

Имеются два термина, часто используемых, когда речь заходит об объектно-ориентированных языках программирования: раннее и позднее связывание. По отношению к C++ эти термины соответствуют событиям, которые возникают на этапе компиляции и на этапе исполнения программы соответственно.

В терминах объектно-ориентированного программирования раннее связывание означает, что объект и вызов функции связываются между собой на этапе компиляции. Это означает, что вся необходимая информация для того, чтобы определить, какая именно функция будет вызвана, известна на этапе компиляции программы. В качестве примеров раннего связывания можно указать стандартные вызовы функций, вызовы перегруженных функций и перегруженных операторов. Принципиальным достоинством раннего связывания является его эффективность — оно более быстрое и обычно требует меньше памяти, чем позднее связывание. Его недостатком служит невысокая гибкость.

Позднее связывание означает, что объект связывается с вызовом функции только во время исполнения программы, а не раньше. Позднее связывание достигается в C++ с помощью использования виртуальных функций и производных классов. Его достоинством является высокая гибкость. Оно может использоваться для поддержки общего интерфейса, позволяя при этом различным объектам иметь свою собственную реализацию этого интерфейса. Более того, оно помогает создавать библиотеки классов, допускающие повторное использование и расширение.

Какое именно связывание должна использовать программа, зависит от предназначения программы. Фактически достаточно сложные программы используют оба вида связывания. Позднее связывание является одним из самых мощных добавлений языка C++ к возможностям языка C. Платой за такое увеличение мощи программы служит некоторое уменьшение ее скорости исполнения. Поэтому использование позднего связывания оправдано только тогда, когда оно улучшает структурированность и управляемость программы. Надо иметь в виду, что проигрыш в производительности невелик, поэтому когда ситуация требует позднего связывания, можно использовать его без всякого сомнения.

Механизм позднего связывания

Связывание — это сопоставление вызова функции с вызовом.

Рассмотрим механизм, осуществляющий динамическое связывание. В традиционном C вызов функции происходит при помощи так называемого механизма раннего или статического связывания. Компилятор генерирует код вызова заранее известной конкретной функции по ее имени, и при сборке этот вызов связывается с реализацией функции. При раннем связывании адрес вызываемой функции всегда известен во время сборки программы.

Но подобный механизм раннего связывания не всегда позволяет легко добавлять к имеющимся типам данных новые наследуемые типы, использующие собственные реализации унаследованных функций. Проблема состоит в том, что если добавить наследуемый тип данных, в котором переопределяются некоторые функции, то в этом случае нельзя использовать старый код программы для их вызова. Когда создавалась программа, она еще “не знала” о том, что в дальнейшем появятся какие-то новые функции, к которым ей придется обращаться. И даже если при выполнении каких-то действий с объектом нового типа программа, по логике, должна обратиться к переопределенной функции унаследованного типа, все равно произойдет вызов базовой функции типа, так как именно эта функция жестко привязана к конкретному коду программы.

Это приводит к тому, что при добавлении новых типов приходится изменять код программы, который обращается к функциям, переопределенным в классах потомках, и затем проводить повторную компиляцию всей программы.

Механизм позднего связывания позволяет передавать сообщение о вызове виртуальной функции тому объекту, над которым выполняется действие. При этом виртуальная функция не связывается с объектом так, как это происходит с обычными функциями в процессе компиляции. Можно создать новый класс, унаследованный от базового, и написать для него другую подходящую функцию, соответствующую виртуальной функции базового типа. Затем можно скомпилировать свой код и скомпоновать его с уже готовыми объектными файлами кода функций, обращающихся к виртуальным функциям объектов базового типа. Вызов необходимой функции (для объектов базового или унаследованного типа) осуществляется уже в процессе выполнения программы и полностью зависит от типа объекта, участвующего в вызове.

Итак, адрес функции, соответствующей данному объекту, определяется во время выполнения программы, что именно и характеризует понятие позднего связывания. Во время компиляции в C++ осуществляется проверка того, что функция определена, и аргументы и возвращаемое значение имеют верный тип. Однако в момент компиляции неизвестно, какой именно код будет выполняться при вызове виртуальной функции. Поэтому на место реального вызова функции компилятор помещает в программу несколько специальных операторов.

В процессе работы программы эти операторы извлекают указатель, используемый для вычисления адреса нужной функции, из самого объекта. По этому указателю, который в C++ называется VPTR (Virtual PoinTeR), из специальной таблицы выбирается физический адрес вызываемой функции. Эта таблица, называемая в C++ VTABLE, содержит адреса всех виртуальных функций. Каждый класс, имеющий или унаследовавший виртуальные функции, имеет и соответствующую ему таблицу VTABLE. Любой объект имеет свой указатель VPTR. При вызове виртуальной функции самим объектом определяется, какой код должен выполняться.